User manual for SHEBO
Surrogate optimization of computationally expensive black-box problems with hidden constraints

Juliane Müller (JulianeMueller@lbl.gov)
Lawrence Berkeley National Laboratory
Center for Computational Sciences and Engineering

This user manual accompanies the implementation of the SHEBO code that was developed for solving computationally expensive black-box optimization problems whose objective function may fail to return a value for some parameter vectors. These failures are not necessarily due to the simulation crashing, but rather due to, for example, underlying solvers not converging or other constraints that cannot be assessed for being satisfied beforehand. In the literature, these constraints are often referred to as "hidden" constraints. The code accompanies our paper **"Surrogate Optimization of Computationally Expensive Black-Box Problems with Hidden Constraints" by Juliane Müller and Marcus Day that was published in the *Informs Journal on Computing*, in 2019 (doi: https://doi.org/10.1287/ijoc.2018.0864)**. We recommend reading this article to understand the underlying assumptions and the workings of the algorithm.

The accompanying code is available on bitbucket: https://bitbucket.org/julianem/shebo-hidden-constraint-optimization/src/master/.

Obtaining the code

Go to the bitbucket repository (https://bitbucket.org/julianem/shebo-hidden-constraint-optimization/src/master/) and clone/download the repo to your local machine.

Prerequisites and dependencies

The code was developed and tested on Linux using Python 2.7. A Python 3 version does not yet exist. In addition, the code calls the third-party optimizer NOMAD for doing the local search. It relies on NOMAD version 3.9.1 (see https://www.gerad.ca/nomad/). Download NOMAD and follow its installation instructions. We recommend that the user try a basic single objective batch example with their NOMAD version to ensure it runs.
Be sure you have the following python packages installed:

- numpy
- cPickle
- scipy
- subprocess
- math
- shutil

Adjust directory paths and objective function

Next, in your local hidden_constraints folder where you downloaded the SHEBO code to, make a directory called tmp.
There some files that have to be altered by the user (adjustment of search paths). These are :

- In basic_params.txt: adjust the directory defined for TMP_DIR to your local, just created tmp directory path.
- In objective.py: This is the definition of the objective function. Currently, a simple placeholder function is used that "fails" to evaluate if the first parameter value is larger than 0.9 (return NaN) and that otherwise returns the sum of the squares of the parameter values. It is important that the returned value (y) of *your own objective function* is a scalar. Note that it is assumed that all parameters live in [0,1]. If your problem's upper and lower parameter bounds are different, you have to rescale them inside of this function as shown in the comments in this function. Moreover, the path of my_in must be adjusted to your local directory.
- In objectivewrap_for_nomad.py: Adjust the two directory paths to your own paths.
- In write_nomad_prms.py: Adjust all paths to directories to your local directories of the SHEBO code and the NOMAD installation, respectively.

Running the code with the simple placeholder objective

Before trying it on your expensive simulation code, make sure it program runs for this simple provided objective function (sum of squares). In a terminal, maneuver to your SHEBO directory and run
python hico_optimizer.py
This should lead to output starting with
running hidden constraints code
followed by information about iteration number, number of function evaluations and so on. Once the program finishes, it will write an output file with the name result_test_4.data. To look at its contents, start python, and do
>>> import cPickle as p
>>> s=p.load(open("result_test_4.data"))
>>> s.Y (this shows the function values done)
>>> s.S (this shows the parameter vectors that were evaluated)
>>> s.xbest (this shows the best parameter vector found (in this case a vector of zeros))
>>> s.fbest (this shows the best objective function value found)

There is a short script that plots a progress plot (best objective function value versus the number of function evaluations. Type in the terminal:
python analyze_results.py (make sure the file loaded here has the same name as the output file that was created). This should give you a progress plot similar to the one in progress.png (see Figure 1 below).

Once you have run the code successfully, you should alter objective.py such that it calls your expensive simulation.

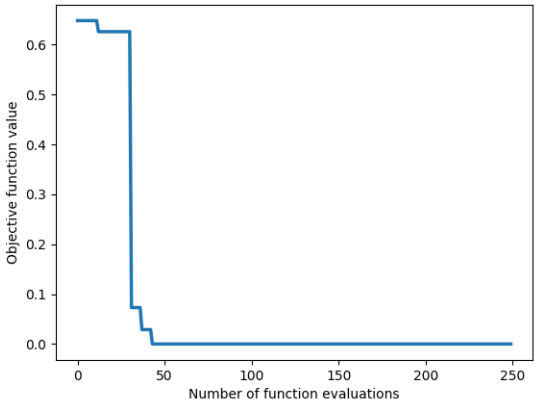

Figure 1: Progress plot for the simple test example.